

Amendments to the Specification:

Please replace the paragraph beginning at page 1, line 4
with the following amended paragraph:

~~Jini~~-(TM) JiniTM is an established specification which
allows application services to announce their availability
dynamically. Users of services can dynamically find an
application service using JiniTM. Examples can include services
providing freight rates, stock quotes, price inventory levels
for products or modules (e.g., calculators) for applications.
~~Jini~~-(TM) JiniTM operates by registering services, and responding
to client inquiries about one of these services. ~~Jini~~-(TM)
JiniTM also provides proxy code to the client allowing the client
to communicate with the services. In this way, ~~Jini~~-(TM) JiniTM
acts as a broker between the client and the existing services.

Please replace the paragraph beginning at page 1, line 15 with the following amended paragraph:

~~Jini~~-(TM) JiniTM has been written in a way such that it can only be used by services written in JavaTM. Many non-JavaTM services ("legacy" services), however exist. These services cannot operate through ~~Jini~~-(TM) JiniTM.

Please replace the paragraph beginning at page 2, line 1 with the following amended paragraph:

Figure 1 shows an operation of the bridge of the present system, forming an interface between non Java services and a ~~Jini~~-(TM) JiniTM broker;

Please replace the paragraph beginning at page 2, line 5 with the following amended paragraph:

Figures 3A and 3B show[[s]] an example of encapsulating CORBA component.

Please replace the paragraph beginning at page 2, line 9 with the following amended paragraph:

The present application defines a bridge between non JavaTM services and a client which is requesting such services. More

specifically, the bridge provides a code generator to generate a proxy code that wraps the non-JavaTM object, and hence enables communication between the client and the service. A surrogate object publishes information about the service and proxy. The surrogate object maintains this information using the ~~Jini-(TM)~~ JiniTM protocol. In this way, ~~Jini-(TM)~~ JiniTM can be used in its normal way, but can be used as a broker for non-JavaTM services. JiniTM stores the code, by indexing the code directly, or by indexing proxy code.

Please replace the paragraph beginning at page 2, line 20 with the following amended paragraph:

The current ~~Jini-(TM)~~ JiniTM interface will register services which have a JavaTM Application Programming Interface ("API"). Services without a JavaTM API cannot be registered with ~~Jini-(TM)~~ JiniTM. While the present application specifically describes using ~~Jini-(TM)~~ JiniTM, it should be understood that any service requiring a JavaTM API could be similarly used.

Please replace the paragraph beginning at page 3, line 3 with the following amended paragraph:

The operation is shown in figure 1. A JavaTM service normally publishes itself or its proxy to ~~Jini~~^(TM) JiniTM, along with a set of search attributes that can be used by services that are searching among the services. According to the present system, a non JavaTM service 100 is published, using a JavaTM compliant "wrapper", as described. The wrapper acts like Java proxy code, and points to the non JavaTM service, in the same way that JavaTM proxy code would point to the JavaTM service.

Please replace the paragraph beginning at page 3, line 11 with the following amended paragraph:

A GUI (Graphical User Interface) allows a user to point and click the location of a service interface file representing the non-JavaTM component, here an ActiveX component. For example, this may be an OCX file for an activeX component 100 as shown in figure 1. The point and click operation is used to instruct the bridge 110 to publish the service.

Please replace the paragraph beginning at page 4, line 3 with the following amended paragraph:

At 230, this system forms middle tier tunneling proxy code 140 to form a bridge between the client and the service. Other

information obtained from introspection at 210 is used to generate search attributes. These form keywords which are used to supplement the keyword repository in ~~Jini~~^(TM) JiniTM. In addition to the keywords identified by introspection, a user may also specify additional keywords in the Graphical User Interface (GUI).

Please replace the paragraph beginning at page 4, line 11 with the following amended paragraph:

~~Jini~~^(TM) JiniTM stores the proxy objects and a set of search attribute objects shown as 152, 154, shown in figure 1.

Please replace the paragraph beginning at page 4, line 13 with the following amended paragraph:

The ~~Jini~~^(TM) JiniTM service is shown as 150. A client, or a service-proxy for the client, makes a call to the backend service wrapper object. The wrapper object redirects the call to the actual Component Object Model ("COM") or CORBA component. Each time such a call is made, the bridge 110 generates code that redirects the calls to the service 100. The generated code may be JavaTM proxy code.

Please replace the paragraph beginning at page 4, line 19 with the following amended paragraph:

At 260, the bridge receives a ~~Jini~~-(TM) JiniTM lease object based on the successful registration of the ~~Jini~~-(TM) JiniTM object. The lease object shown as 142 keeps the object definition up-to-date. The bridge renews the lease from time to time, thereby insuring up-to-date information in ~~Jini~~-(TM) JiniTM.

Please replace the paragraph beginning at page 5, line 1 with the following amended paragraph:

A service is published to ~~Jini~~-(TM) JiniTM in the following way. First, the service is serialized as a JavaTM object, if it can be so serialized. Other, non JavaTM compliant services are packaged as described above. Services which are incapable of being serialized into ~~Jini~~-(TM) JiniTM may publish their proxies instead. Constraints such as heaviness, native dependability, and the like may prevent the service from directly publishing to ~~Jini~~-(TM) JiniTM. The proxy is a serializable lightweight JavaTM object that acts as the service delegate. The proxies act as middle tiered objects allowing access of services on the back end. The object or proxy is sent to ~~Jini~~-(TM) JiniTM, along with

a set of search attributes. Once the Java™ objects have been transported to ~~Jini(TM)~~ Jini™, they are delivered to clients responsive to notifications of service matches.

Please replace the paragraph beginning at page 5, line 15 with the following amended paragraph:

Another aspect is to allow publishing either Jini™ or non Jini™ services. Any service that is capable of serialized in itself, and publishing itself to ~~Jini(TM)~~ Jini™, would do so directly. This enables clients to directly use the service objects.

Please replace the paragraph beginning at page 5, line 20 with the following amended paragraph:

Non ~~Jini(TM)~~ Jini™ services may not have a proxy, or such a proxy may need to be generated as part of the ~~Jini(TM)~~ Jini™ registration process.

Please replace the paragraph beginning at page 6, line 4 with the following amended paragraph:

As an example of operation, figure 3 shows how the system could publish and obtain different information. A service 300,

e.g., a CORBA service is coupled to the code generator 305 which provides an wrapper around the CORBA shown as 310. Once in the wrapper, the CORBA code appears to the broker or proxy as Java™ code, i.e., it is no different like any other Java™ application. At 315, the application is published with the ~~Jini~~™ Jini™ broker. Figure 3B, shows a client, which is a non Java™ client such as an Excel™ client asking for services. At 330, the request for services is also placed in a wrapper at 335 and placed to ~~Jini~~™ Jini™. ~~Jini~~™ Jini™ returns the request.